

(/apps
/redirect?url
banner-click

java设计模式 - 组合模式(合成模式 Composite)



步积 (/u/4aec87ce0f2b) +关注

0.3 2017.07.04 20:57* 字数 1442 阅读 353 评论 1 喜欢 3

(/u/4aec87ce0f2b)

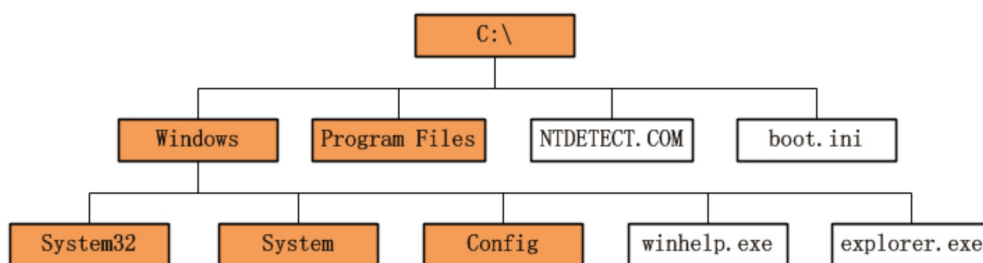
定义

属于对象的结构模式，有时又叫做“部分——整体”模式。组合模式将对象组织到树结构中，可以用来描述整体和部分的关系。组合模式可以使客户端将单纯元素与复合元素同等看待。

组合模式

组合模式把部分和整体的关系用树结构表示出来。组合模式使得客户端把一个个单独的成分对象和由它们复合而成的复合对象同等看待。

例如：一个文件系统就是典型的组合模式系统。下面是常见的Windows文件系统的一部分。



文件系统的组合模式

从上图可以看出，文件系统是一个树结构，树上长有节点。树的节点有两种，一种是树枝节点，也就是目录，有内部树结构，在图中涂有颜色；另一种是树叶节点，也就是文件，没有内部树结构。

显然，可以把目录和文件当做同一种对象同等对待和处理，这也就是组合模式的应用。

组合模式可以不提供父对象的管理方法，但是组合模式必须在合适的地方提供子对象的管理方法，诸如：`add()`、`remove()`、以及`getChild()`等。

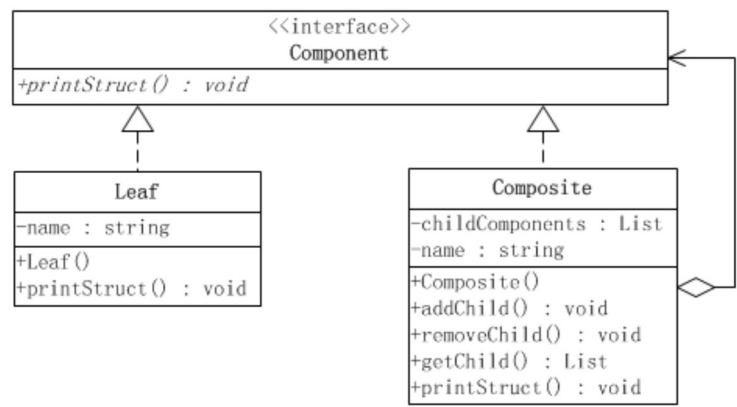


组合模式的实现根据所实现的借口的区别分为两种形式，分别称为**安全式**和**透明式**。

安全式组合模式的结构

安全模式的组合模式要求管理聚集的方法只出现在树枝构件类中，而不出现在树叶构件类中。

(/apps
/redirect?ui
banner-clip



安全式的组合模式

这种形式涉及到三个角色：

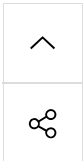
- **抽象构件(Component)角色**：这是一个抽象角色，它给参加组合的对象定义出公共的接口及其默认行为，可以用来管理所有的子对象。组合对象通常把它所包含的子对象当做类型为 Component 的对象。在安全式的组合模式里，构建角色并不定义出管理子对象的方法，这一定义由树枝构件对象给出。
- **树叶构件(Leaf)角色**：树叶对象没有下级子对象的对象，定义出参加组合的原始对象的行为。
- **树枝构件(Composite)角色**：代表参加组合的有下级子对象的对象。树枝构件类给出所有的管理子对象的方法，如 add()、remove()、以及 getChild() 等。

示例代码

抽象构件角色类

```
public interface Component {  
    /**  
     * 输出组件自身的名称  
     * @param preStr 前缀  
     */  
    public void printStruct(String preStr);  
}
```

树枝构件角色类



```
public class Composite implements Component {
    /**
     * 用来存储组合对象中包含的子组件对象
     */
    private List<Component> childComponents = new ArrayList<Component>();
    /**
     * 组合对象的名称
     */
    private String name;
    /**
     * 构造方法，传入组合对象的名称
     * @param name 组合对象的名称
     */
    public Composite(String name) {
        this.name = name;
    }

    /**
     * 聚集管理方法，增加一个子构建对象
     * @param child 子构建对象
     */
    public void addChild(Component child) {
        this.childComponents.add(child);
    }

    /**
     * 聚集管理方法，删除一个子构建对象
     * @param index 子构建对象的下标
     */
    public void removeChild(int index) {
        this.childComponents.remove(index);
    }

    /**
     * 聚集管理方法，返回所有子构建对象
     * @return 子构建对象列表
     */
    public List<Component> getChild() {
        return this.childComponents;
    }

    /**
     * 输出对象的自身结构
     * @param preStr 前缀，主要是按照层级拼装空格，实现向后缩进
     */
    @Override
    public void printStruct(String preStr) {
        //首先输出自身
        System.out.println(preStr + '+' + this.name);

        //如果还包含有子组件，那么就输出这些子组件对象
        if (this.childComponents != null) {
            //添加前缀空格，表示向后缩进
            preStr += " ";

            //循环递归输出每个子对象
            for (Component component : childComponents) {
                component.printStruct(preStr);
            }
        }
    }
}
```

(/apps
/redirect?url
banner-clip



```
}  
  
}
```

树叶构件角色类

(/apps
/redirect?url
banner-click

```
public class Leaf implements Component {  
    /**  
     * 叶子对象的名称  
     */  
    private String name;  
    /**  
     * 构造方法，传入叶子对象的名称  
     * @param name 叶子对象的名称  
     */  
    public Leaf(String name) {  
        this.name = name;  
    }  
    /**  
     * 输出叶子对象，因为叶子对象没有子对象，也就是输出叶子对象的名称。  
     * @param preStr 前缀，主要是按照层级进行拼接的空格，用于实现向后缩进  
     */  
    @Override  
    public void printStruct(String preStr) {  
        System.out.println(preStr + "-" + name);  
    }  
}
```

客户端类

```
public class Client {  
    public static void main(String[] args) {  
        Composite root = new Composite("服装");  
        Composite c1 = new Composite("男装");  
        Composite c2 = new Composite("女装");  
  
        Leaf leaf1 = new Leaf("衬衫");  
        Leaf leaf2 = new Leaf("夹克");  
        Leaf leaf3 = new Leaf("裙子");  
        Leaf leaf4 = new Leaf("套装");  
  
        root.addChild(c1);  
        root.addChild(c2);  
        c1.addChild(leaf1);  
        c1.addChild(leaf2);  
        c2.addChild(leaf3);  
        c2.addChild(leaf4);  
  
        root.printStruct("");  
    }  
}
```

可以看出，树枝构件类 Composite 给出了 add()、remove()、以及 getChild() 等方法的声明和实现，而树叶构件类则没有给出这些方法的声明或实现。这样的做法是安全的做法，由于这个特点，客户端应用程序不可能错误的调用树叶构件的聚集方法，因为树叶

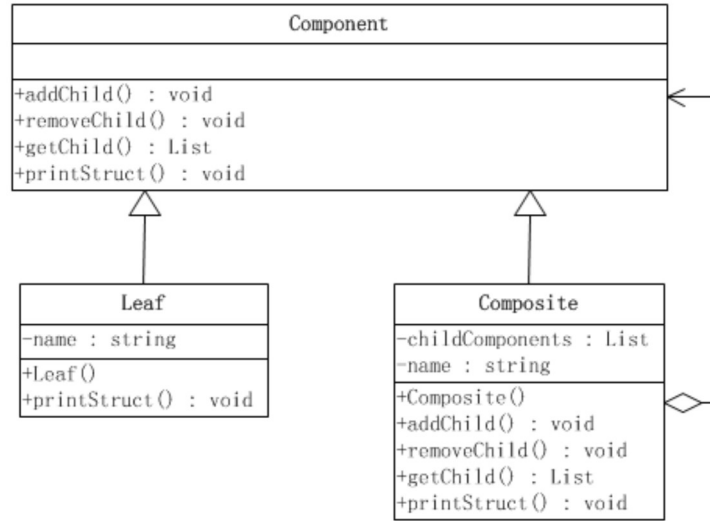


构件没有这些方法，调用会导致编译错误。

安全式组合模式的缺点是不够透明，因为树叶类和树枝类将具有不同的接口。

透明式组合模式的结构

与安全式的组合模式不同的是，透明式的组合模式要求所有的具体构建类，不论树枝构件还是树叶构件，均符合一个固定接口。



透明式的组合模式

示例代码

抽象构件角色类

(/apps
/redirect?url
banner-clip



```
public abstract class Component {  
    /**  
     * 打印组件自身的名称  
     * @param preStr 前缀，用于实现缩进  
     */  
    public abstract void printStruct(String preStr);  
  
    /**  
     * 聚集管理方法，增加一个子构件对象  
     * @param child 子构件对象  
     */  
    public void addChild(Component child) {  
        /**  
         * 缺省实现，抛出异常，因为树叶对象没有此功能  
         * 或者子组件没有实现这个功能  
         */  
        throw new UnsupportedOperationException("对象不支持此功能");  
    }  
  
    /**  
     * 聚集管理方法，删除一个子构件对象  
     * @param index 子构件对象的下标  
     */  
    public void removeChild(int index) {  
        /**  
         * 缺省实现，抛出异常，因为树叶对象没有此功能  
         * 或者子组件没有实现这个功能  
         */  
        throw new UnsupportedOperationException("对象不支持此功能");  
    }  
  
    /**  
     * 聚集管理方法，返回所有子构件对象  
     * @return 返回所有子构件对象  
     */  
    public List<Component> getChild() {  
        /**  
         * 缺省实现，抛出异常，因为树叶对象没有此功能  
         * 或者子组件没有实现这个功能  
         */  
        throw new UnsupportedOperationException("对象不支持此功能");  
    }  
}
```

(/apps
/redirect?u
banner-clip

树枝构件角色类，此类将 `implements Component` 改为 `extends Component`，其他地方无变化。



```
public class Composite extends Component {
    /**
     * 用来存储对象中包含的子构件对象
     */
    private List<Component> childComponents = new ArrayList<Component>();

    /**
     * 组合对象的名称
     */
    private String name;

    /**
     * 构造方法，传入组合对象的名称
     * @param name 组合对象的名称
     */
    public Composite(String name) {
        this.name = name;
    }

    /**
     * 聚集管理方法，增加一个子构件对象
     * @param child 子构件对象
     */
    public void addChild(Component child) {
        childComponents.add(child);
    }

    /**
     * 聚集管理方法，删除一个子构件对象
     * @param index 子构件对象的下标
     */
    public void removeChild(int index) {
        childComponents.remove(index);
    }

    /**
     * 聚集管理方法，返回所有子构件对象
     * @return 返回所有子构件对象
     */
    public List<Component> getChild() {
        return childComponents;
    }

    public void printStruct(String preStr) {
        //首先输出自身
        System.out.println(preStr + '+' + this.name);

        //如果还包含有子组件，那么就输出这些子组件对象
        if (this.childComponents != null) {
            //添加前缀空格，表示向后缩进
            preStr += "  ";

            //循环递归输出每个子对象
            for (Component component : childComponents) {
                component.printStruct(preStr);
            }
        }
    }
}
```

(/apps
/redirect?url
banner-clip



树叶构件角色类，此类将 `implements Component` 改为 `extends Component`，其他地方无变化。

```
public class Leaf extends Component {
    /**
     * 组合对象的名称
     */
    private String name;

    /**
     * 构造方法，传入组合对象的名称
     * @param name 组合对象的名称
     */
    public Leaf(String name) {
        this.name = name;
    }

    /**
     * 输出叶子对象，因为叶子对象没有子对象，也就是输出叶子对象的名称。
     * @param preStr 前缀，主要是按照层级进行拼接的空格，用于实现向后缩进
     */
    @Override
    public void printStruct(String preStr) {
        System.out.println(preStr + "-" + name);
    }
}
```

(/apps
/redirect?url
banner-clip

客户端类的主要变化是不再区分树枝构件角色 `Composite` 对象和树叶构件角色 `Leaf` 对象。

```
public class Client {
    public static void main(String[] args) {
        Component root = new Composite("服装");
        Component c1 = new Composite("男装");
        Component c2 = new Composite("女装");

        Component leaf1 = new Leaf("衬衫");
        Component leaf2 = new Leaf("夹克");
        Component leaf3 = new Leaf("裙子");
        Component leaf4 = new Leaf("套装");

        root.addChild(c1);
        root.addChild(c2);
        c1.addChild(leaf1);
        c1.addChild(leaf2);
        c2.addChild(leaf3);
        c2.addChild(leaf4);

        root.printStruct("");
    }
}
```

可以看出，客户端无需再区分操作的是树枝对象还是树叶对象了，对于客户端而言，操作的都是 `Component` 对象。



两种实现方式的选择

这里所说的安全式组成模式是指：从客户端使用组成模式上看是否更安全，如果是安全的，那么就不会有发生误操作的可能，能访问的方法都是被支持的。

这里所说的透明性组成模式是指：从客户端使用组成模式上，是否需要区分到底是“树枝对象”还是“树叶对象”。如果是透明的，那就不用区分，对于客户而言，都是 Component 对象，具体的类型对于客户端而言是透明的，是无需关心的。

对于组合模式而言，在安全性和透明性上，会**更看重透明性**，毕竟组合模式的目的是：让客户端不再区分操作的是树枝对象还是树叶对象，而是以一个统一的方式来操作。

而且对于安全性的实现，需要区分的是树枝对象还是树叶对象。有时候，需要将对象进行类型转换，却发现类型信息丢失了，只好强行转换，这种类型转换必然是不够安全的。

因此在使用组合模式的时候，建议多采用透明式的实现方式。

参考

《JAVA与模式》之合成模式 (<https://link.jianshu.com?t=http://www.cnblogs.com/java-my-life/archive/2012/04/17/2453861.html>)

如果我的文章对你有帮助，可以打赏一下，请我吃块糖咯

赞赏支持

📖 JAVA面试 (/nb/11235220)

举报文章 © 著作权归作者所有



步积 (/u/4aec87ce0f2b) ♂

写了 363087 字，被 479 人关注，获得了 636 个喜欢

(/u/4aec87ce0f2b)

+ 关注

走在面试的路上，基础学起，搞定大厂

喜欢 | 3



更多分享





(/apps/redirect?utm_source=note-bottom-click)

(/apps
/redirect?u
banner-lic



登录 (/sign-in?utm_source=desktop&utm_medium=not-signed-in-comr

1条评论

只看作者

按时间倒序 按时间正序



Jason_陈 (/u/192d11df754c)

2楼 · 2019.05.07 10:25

(/u/192d11df754c)

很棒! 暂时明白了, 您能否讲一下这个模式适用于哪些场景呢? 或者举个例子

赞 回复

被以下专题收入, 发现更多相似内容



java技术基础 (/c/4c04f6a0ad2a?utm_source=desktop&utm_medium=notes-included-collection)



Java学习笔记 (/c/04cb7410c597?utm_source=desktop&utm_medium=notes-included-collection)

(/p/dead42334033?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

【结构型模式十】组合模式(Composite) (/p/dead4233...

1 场景问题# 1.1 商品类别树## 考虑这样一个实际的应用：管理商品类别树。在实现跟商品有关的应用系统的时候，一个很常见的功能就是商品类别树的管理，比如有如下所示的商品类别树：仔细观察上面的商品类别树，有以下几个明显的特点：有一个根节点，比如服装，它没有父节点，它...

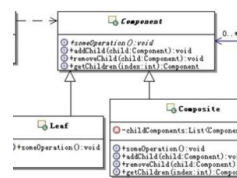


图 15.1 组合模式结构示意图




猿码道 (/u/657c611b2e07?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

设计模式汇总 (/p/f7f9553ba2ba?utm_campaign=maleskine&utm_conte...

设计模式汇总 一、基础知识 1. 设计模式概述 定义：设计模式（Design Pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结，使用设计模式是为了可重用代码、让代码更容易被他人理解并且保证代码可靠性。根据它们的用途，设计模式可分为 创建型...

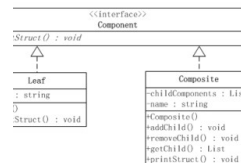


 MinoyJet (/u/9c0529da1861?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


(/p/94208b5fc4bc?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

Java组合模式 (/p/94208b5fc4bc?utm_campaign=ma...

概念 合成模式属于对象的结构模式，有时又叫做“部分——整体”模式。合成模式将对象组织到树结构中，可以用来描述整体与部分的关系。合成模式可以使客户端将单纯元素与复合元素同等看待。合成模式 合成模式把部分和整体的关系用树结构表示出来。合成模式使得客户端把一个个单独的成分对象和...



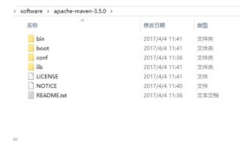
(/apps
/redirect?u
banner-clip


 今晚打肉山 (/u/b1644eecac2a?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/a3be274fc60e?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

设计模式--组合模式 (/p/a3be274fc60e?utm_campaig...

目录 本文的结构如下： 引言 什么是组合模式 模式的结构 典型代码 代码示例 优点和缺点 适用环境 模式应用 一、引言 树形结构是很常见的，比如目录系统，随便点开一个文件夹，文件夹下面可能有文件，也有子文件夹，子文件夹中还有子子文件夹和文件..... 还有导航中的菜单。 ...




 w1992wishes (/u/44304509b1eb?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/b794f23ad296?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

找个时间 (/p/b794f23ad296?utm_campaign=maleski...


熬周记1639：找个时间 原创2016-09-25@草根公务员的自留地 周一一篇，碎碎念..... 【本周事】 皮特和朱莉离婚了，作为《Friends》的忠实拥趸，我觉得布拉德·皮特还是和詹尼佛·安妮斯顿更配。（这么大岁数，还那么八卦，恶心！） 【记录】 前几天，刚刚激情满怀地参...



 老土 (/u/b36e08462ff1?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

菜妹子 (20) (/p/55be7360042f?utm_campaign=maleskine&utm_conte...

轮到新媳妇过来，那可就更热闹了，挡道的人也就更多了：本姓里的嫂嫂、小姑、小叔们都挡道，还有爱热闹的邻家嫂嫂辈的都来凑哄伙。两位娘家送亲的婆姨将新媳妇轻轻的扶下马，左右搀着小心翼翼的走上前来要求放行。这时，有巧嘴的婆姨就说一些顺口溜，让新媳妇跟着学舌。不学不让过。这些顺口溜大...


 shq海琼 (/u/d022bc721931?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)



(/p/28d033c93222?utm_campaign=maleskine&
utm_content=note&utm_medium=seo_notes&
utm_source=recommendation)

市井 (/p/28d033c93222?utm_campaign=maleskine...



 御风而行_2c6f (/u/0d97095820f7?utm_campaign=maleskine&
utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


(/apps
/redirect?ui
banner-clic

(/p/c923dfc8555e?utm_campaign=maleskine&
utm_content=note&utm_medium=seo_notes&
utm_source=recommendation)

【济宁交警支队创城简报】 (/p/c923dfc8555e?utm_c...

4月30日，白天到夜间，全市天气晴间多云，南风，19~31℃。截止4月30日8时，济宁城区、国省道路况良好，通行正常。【重点工作】4月29日上午，市局党委委员、交警支队长王泽劲，支队党委委员、副支队长李华在城市调研督导各单位文明城市创建工作开展情况。支队党委副书记、政...




 济宁交警指挥中心 (/u/588fc57eaae5?utm_campaign=maleskine&
utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/9593f01401cd?utm_campaign=maleskine&
utm_content=note&utm_medium=seo_notes&
utm_source=recommendation)

为什么签合同要盖章，不妨看看公章发展史 (/p/9593f0...

公章，古称官印，是行使权力的信物，故也称为“印信”，它作为中国传统文化的一部分，有着悠久的历史渊源和发展历程。从古书记载来看，官印的出现可以追溯到四千多年前的尧、舜时代。当然，比较完备的官印制度，形成于秦代，自丞相太尉到郡守县令，都由国君在任命时授予官印，同时配发穿在印钮...



 陈若缺 (/u/86572a9d4880?utm_campaign=maleskine&utm_content=user&
utm_medium=seo_notes&utm_source=recommendation)

